University of Saskatchewan
Department of Computer Science

Cmpt 330
Final Examination

December 12, 2003

Time: 3 hours
Total Marks: 97

Professor: A. J. Kusalik
Closed Book†

Name: _____

Student Number: _____

Directions:

Answer each of the following questions in the space provided in this exam booklet. If you must continue an answer (e.g. in the extra space on the last page, or on the back side of a page), make sure you clearly indicate that you have done so and where to find the continuation. Ensure that all answers are written legibly; no marks will be given for answers which cannot be deciphered. Where a discourse or discussion is called for, please be concise and precise. Do not give "extra answers". Extra answers which are incorrect may result in your being docked marks. If any question requires an assumption as to a particular operating system context, assume UNIX as manifest by NetBSD V1.6. If you find it necessary to make any other assumptions to answer a question, state the assumption with your answer.

Marks for each major question are given at the beginning of that question. Supplemental information, in the form of excerpts from possibly useful *man* pages, is at the end of the exam booklet.

Calculators may be used for this exam. However, you should not need them.

Good luck.

For marking use only:

A.	_____/7	D.	_____/14	G.	_____/11
B.	_____/10	E.	_____/10	H.	_____/6
C.	_____/5	F.	_____/24	I.	_____/10

Total: _____/97

† Closed book, except for one optional 8.5×11 inch quick reference sheet ("cheat sheet") of the student's own compilation.

A. (7 marks)

Give the corresponding expansion for each of the following acronyms or contractions:

1. DNS

2. fsck

3. IPC

4. mknod

5. NTP

6. SIGCHLD

7. UDP

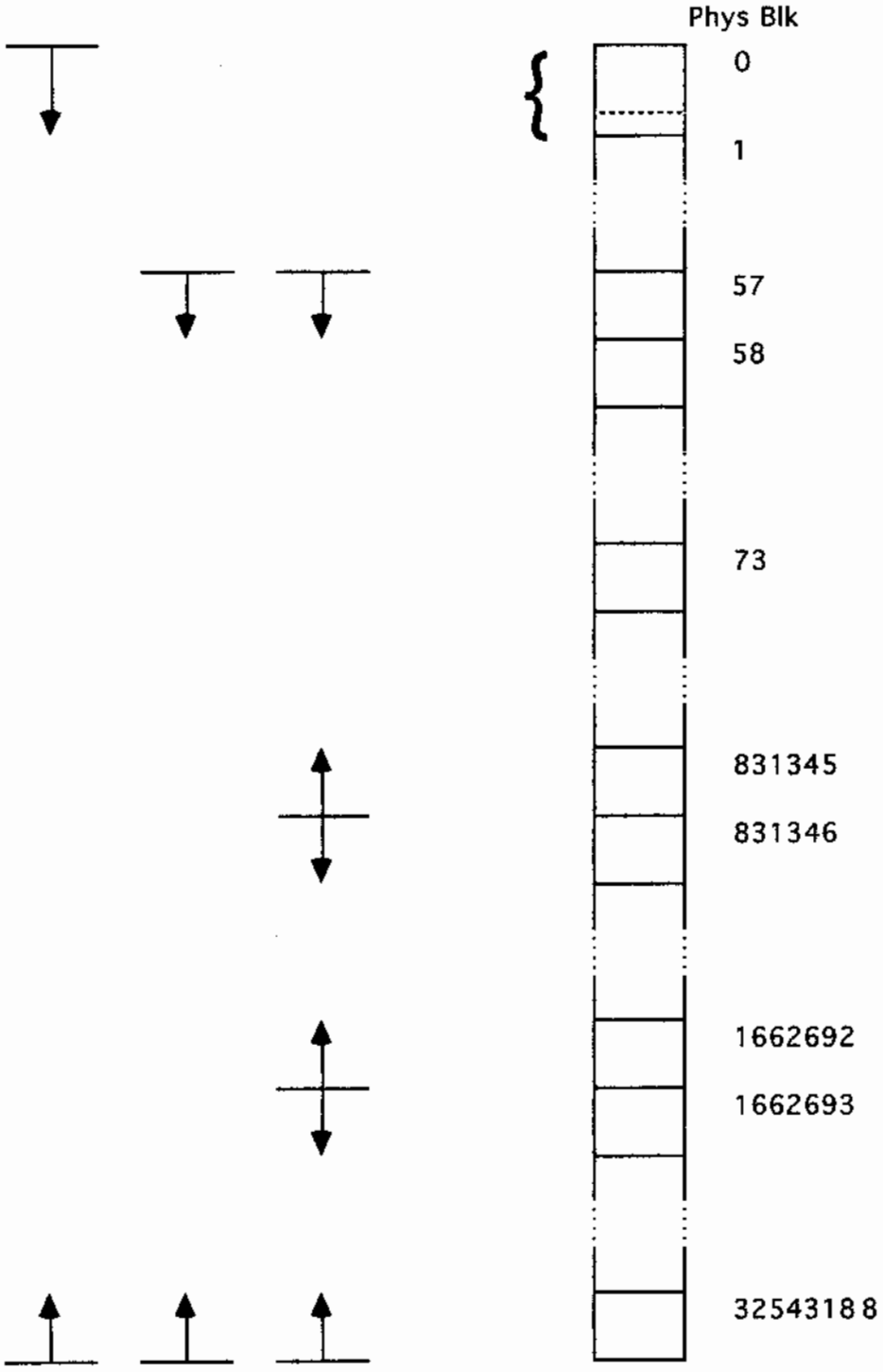
B. (10 marks)

The following diagram shows the partitioning of a physical disk on a computer with a PC architecture running NetBSD V1.6. Assume the disk has 57 sectors per track and is bootable. Label the diagram using labels drawn only following set of potential labels:

backup superblock	BIOS partition table
block 831346 of partition c	duplicate BIOS partition table
free-list reserve	MBR
NetBSD partition table	primary UNIX bootstrap
superblock of UNIX file system	UNIX partition a
UNIX partition b	UNIX partition c
UNIX partition d	UNIX partition e

Follow standard UNIX and NetBSD conventions regarding use of particular partition names.

Note that there are extra labels in the set above; i.e. some of the labels are not appropriate for the diagram and should not be used.


Phys Blk
0
1
57
58
73
831345
831346
1662692
1662693
32543188

C. (5 marks)

Consider the following files dealing with system management, administration, configuration, or operations in a BSD UNIX system:

a. `/dev/MAKEDEV`

b. `/etc/disktab`

c. `/etc/fstab`

d. `/etc/hosts`

e. `/etc/inetd.conf`

f. `/etc/nsswitch.conf`

g. `/etc/passwd`

h. `/etc/protocols`

i. `/etc/services`

j. `/usr/include/netinet/in.h`

k. `/usr/include/sys/signal.h`

l. `/usr/include/sys/socket.h`

m. `/usr/include/sys/un.h`

n. `/usr/share/misc/magic`

Below are excerpts from certain of the files above. For each of the excerpts, indicate which of the files (above) it comes from; i.e. for each of the file samples below, give the label of the file above from which the sample was taken. Note that there are more possible files than there are file excerpts. Therefore, some of the file labels above will not be used below.

1. _____

```
0       string          \037\213        gzip compressed data
>2      byte            <8              \b, reserved method,
>2      byte            8               \b, deflated,
>3      byte            &0x01           ASCII,
>3      byte            &0x02           continuation,
>3      byte            &0x04           extra field,
>3      byte            &0x08           original filename,
>>10    string          x               `%s',
>3      byte            &0x10           comment,
>3      byte            &0x20           encrypted,
```

2. _____

```
std)
      rm -f console drum mem kmem null zero io klog
      mknod console  c 0 0
      mknod drum     c 4 0 ;     chmod 640 drum;     chgrp kmem drum
      mknod kmem     c 2 1 ;     chmod 640 kmem ;    chgrp kmem kmem
      mknod mem      c 2 0 ;     chmod 640 mem ;     chgrp kmem mem
      mknod null     c 2 2 ;     chmod 666 null
      mknod zero     c 2 12 ;    chmod 666 zero
      mknod io       c 2 14 ;    chmod 640 io ;      chgrp kmem io
      mknod klog     c 7 0 ;     chmod 600 klog
```

3. _____

```
ip      0       IP              # internet protocol, pseudo protocol number
icmp    1       ICMP            # internet control message protocol
igmp    2       IGMP            # internet group management protocol
ggp     3       GGP             # gateway-gateway protocol
ipencap 4       IP-ENCAP        # IP encapsulated in IP (officially ``IP'')
st      5       ST              # ST datagram mode
tcp     6       TCP             # transmission control protocol
```

4. _____

```
group:          compat
hosts:          files dns
netgroup:       files [notfound=return] nis
networks:       files
passwd:         compat
passwd_compat:  nis
shells:         files
```

5. _____

```
struct  sockaddr_un {
        u_int8_t  sun_len;              /* total sockaddr length */
        sa_family_t  sun_family;        /* AF_LOCAL */
        char    sun_path[104];          /* path name (gag) */
};
 .
 .
 .
#define SUN_LEN(su) \
        (sizeof(*(su)) - sizeof((su)->sun_path) + \
         strlen((su)->sun_path))
```

D. (2+4+3+1+1+2+1 = 14 marks)

The following two questions require very short, precise answers.

1. What are two names of two of the networks that existed prior to the formation of the Internet, and which were among the set of networks which merged to form the Internet?

2. To execute a command, a shell performs a *fork(2)* call to create a child process, and then the child performs an *exec()* call to start the execution of the desired program. Name four properties or attributes of a process which are inherited across the *exec()* call. I.e. name four process attributes which will stay constant between when the child process calls *exec()* and when the specified program begins execution.

3. Consider the following output produced on one of the machines in the NetBSD (Cmpt330) lab. Then answer the question which follows, based on this output. Note that white space has been added to the output to make it more readable.

```
tonka4# alias ls ls -AF

tonka4# df
Filesystem          1K-blocks     Used    Avail Capacity  Mounted on
/dev/sd0a               31341    21145     8628    71%    /
/dev/sd0d              346389   273765    55304    83%    /usr
procfs                      8        8        0   100%    /proc
barbie:/home           558638   430328   100378    81%    /home
barbie:/usr/pkg        910318   579618   285184    67%    /usr/pkg
barbie:/usr/distrib    910318   579618   285184    67%    /usr/distrib
barbie:/usr/pkgsrc     910318   579618   285184    67%    /usr/pkgsrc

tonka4# ls /usr
X11@      distrib/  install/  libexec/  mdec/     sbin/     tmp/
X11R6/    games/    lib/      lkm/      pkg/      share/
bin/      include/  libdata/  local/    pkgsrc/   src/

tonka4# ls /usr/pkg
bin/      etc/      info/     libdata/  man/      share/
emul/     include/  lib/      libexec/  sbin/     src/

tonka4# ls /usr/pkgsrc
CVS/            cad/            editors/        meta-pkgs/      print/
Makefile        comms/          emulators/      misc/           security/
Packages.txt    converters/     games/          mk/             shells/
README          corba/          graphics/       net/            sysutils/
README.html     cross/          lang/           news/           templates/
archivers/      databases/      mail/           packages@       textproc/
audio/          devel/          math/           pkgtools/       www/
benchmarks/     distfiles/      mbone/          plan9/          x11/

tonka4# ls /usr/distrib
.netrc          NetBSD-1.3.2/   NetBSD-current/ packages/

tonka4# ls /usr/pkgsrc/distfiles
.keep_me                  lesstif-0.86.0.tar.gz     pine4.03.tar.gz
CVS/                      libtool-1.2.tar.gz        texinfo-3.12.tar.gz
bash-2.02.1.tar.gz        nedit_source.tar.gz       vim-5.3-rt.tar.gz
bash-doc-2.02.tar.gz      pine4.02.tar.gz           vim-5.3-src.tar.gz

tonka4# ls -ld /usr/pkgsrc/packages
lrwxr-xr-x  1 root  wheel  19 Sep 24 09:57
 /usr/pkgsrc/packages@ -> ../distrib/packages

tonka4# ls /usr/distrib/packages
1.3/       1.3.2/     README     db/        distfiles@ pkgsrc@

tonka4# ls -ld /usr/distrib/packages/distfiles
lrwxr-xr-x  1 root  wheel  22 Sep 24 09:54
 /usr/distrib/packages/distfiles@ -> ../../pkgsrc/distfiles

tonka4# ls -ld /usr/distrib/packages/pkgsrc
lrwxr-xr-x  1 root  wheel  12 Sep 22 16:50
 /usr/distrib/packages/pkgsrc@ -> ../../pkgsrc
```

Consider the pathname `/usr/pkgsrc/packages/pkgsrc`. This is the pathname of a directory. How many files (ordinary files, subdirectories, etc.) are in that directory (not counting '.' and '..')?

4. Abnormal termination causes a "core file" to be generated. Would you expect to find *data* and *text* areas within that core file? Yes or no?

5. In a device special file, a specific piece of information stored in the inode is actually an index into the device switch table. What is the name for this information?

6. Name any two possible line disciplines for terminals or pseudo-terminals in UNIX.

7. The IP address for a particular host is 192.168.1.254. Is this a class A, B, or C address?

E. (1+1+1+2+3+2 = 10 marks)

Consider the file system that is mounted on `/usr` on one of the machines in the NetBSD lab. Suppose that the *fsdb(8)* program is run on this file system. Part of the transcript of running *fsdb* is as follows:

```
fsdb (inum: 79842)> inode 1908
command `inode 1908
'
current inode: regular file
I=1908 MODE=100555 SIZE=258032
        MTIME=Sep  2 15:30:42 2000 [0 nsec]
        CTIME=Sep  5 07:48:51 2000 [380000000 nsec]
        ATIME=Feb 17 02:14:24 2001 [421130000 nsec]
OWNER=root GRP=wheel LINKCNT=3 FLAGS=0x0 BLKCNT=0x210 GEN=0x1
fsdb (inum: 1908)> blks
command `blks
'
I=1908 33 blocks
Direct blocks:
0: 9488 9496 9504 9512 9520 9528 9536 9544 9552 9560 9568 9576
Indirect block 9584 (level 1):
12: 9592 9600 9608 9616 9624 9632 9640 9648 9656 9664 9672 9680 9688 9696
26: 9704 9712 9720 9728 9736 9744
fsdb (inum: 1908)> quit
```

I.e. *fsdb* was used to "dump" the information in inode 1908. The "blks" command to *fsdb* tells the program to output (as decimal integers) the datablock pointers in the i-node, and in any block pointers pointed to from the inode. Note that the `MODE` field value is in octal, rather than decimal. We know from other commands (such as *dumpfs(8)*) that the datablock size for the file system is 8192 bytes and fragment block size is 1024 bytes. `DEV_BSIZE` for this machine is 512 bytes. Recall that block pointers are in units of fragment blocks; e.g. if a block pointer has value 2, then it is specifying fragment block 2.

1. Is this inode a directory (yes or no)?

2. Is execute permission for "others" turned on for this inode (yes or no)?

3. Does this file contain a hole (yes or no)?

4. What is the size of the fragment?
   Since you don't have calculators, note: 258032/8192 = 31 with 4080 remainder; 258032/1024 = 251 with 1008 remainder; and 258032/512 = 503 with 496 remainder.

5. Byte 0 in physical block 19136 is what byte in the address space of the file? Express your answer in units of K (1K=1024).
   Since you don't have calculators, note: 19136/2 = 9568 and 19136/16 = 1196.

6. Byte 100000 in the address space of the file is stored in which fragment block on the disk?
   And since you don't have calculators: 100000/8192 = 12 with a remainder of 1696, 1696/512 = 3 with a remainder of 160, and 1696/1024 = 1 with a remainder of 672.

F. (3 marks each, total 24 marks)

For each of the following pairs of terms, indicate whether or not they are synonymous (mean the same thing). If they mean different things, contrast the two terms; explain what the two terms mean and how they differ (in meaning). If they are the same, give a definition of the terms. You may use examples to illustrate your point(s).

1. *kernel* and *operating system*

2. In the shell, the *subcommand construct* and the built-in command *exec*

3. *_exit(2)* and *exit(3)*

4. *directory entry* and *hard link*

5. `raise(SIGABRT)` and `abort(3)`

6. *system (CPU) time* and *time-of-day*

7. *network level* in OSI ISO and *internet level* in the TCP/IP architecture

8. *child process* and *client process*

G. (3+3+2+3 = 11 marks)

Answer each of the following questions with a short, precise answer.

1. What is a cylinder group? How would you characterize a cylinder group?

2. Many tape drives provide a block-device interface. Is it practical to support a BSD "fast" file system on such a (tape) device? Why or why not?

3. Why are file names are restricted to 255 characters in BSD UNIX? Base your answer on information in the definition for a `dirent` structure from the file `/usr/include/sys/dirent.h`. That definition is as follows:

```
struct dirent {
    u_int32_t d_fileno;              /* file number of entry */
    u_int16_t d_reclen;              /* length of this record */
    u_int8_t d_type;                 /* file type, see below */
    u_int8_t d_namlen;               /* length of string in d_name */
#define MAXNAMLEN    255
    char    d_name[MAXNAMLEN + 1];   /* name must be no longer than this
*/
};
```

Hint: this could be considered a C programming question.

4. Two of the hallmarks of good software or system design are generality and uniformity. In the case of man-machine interface, therefore, it is advantageous if an operating system can re-use the same concept, and re-use that concept effectively. This is the case in BSD UNIX, where the abstraction of file I/O (achieved via `open()`, `read()`, `write()`, and `close()` system calls) can be used for access to more than just files. Name 3 resources or facilities in BSD UNIX other than ordinary files which are accessed in the same manner as files. Be sure to be clear in your identification of those resources.

H. (3+3=6 marks)

Each of the following questions involves a systems programming or operations scenario. Answer each with a precise and concise answer based on your UNIX systems programming knowledge and experience.

1. A student has composed the answer to a Cmpt330 assignment in a file called `asmnt1.txt`, and wishes to submit the assignment electronically; i.e. wants to send, by email, the contents of `asmnt1.txt` to the instructor. The student uses the following command to submit (e-mail) the assignment:

```
mail -s "Assignment 1 submission" kusalik@cs.usask.ca << asmnt1.txt
```

Unfortunately, and as the student soon finds out, this command is in error. What is wrong with this command? What, in fact, will the command do?

2. Delivery of electronic mail on NetBSD systems is usually handled as follows. A directory, `/var/mail`, contains a file for each user for storing mail received for that user. Thus, there is a file `/var/mail/`*`<username>`* for each user on the system (for each *`<username>`*). For example, on the system `barbie.usask.ca`, there is a file `/var/mail/kusalik` which stores all the mail that has been received for user `kusalik` on machine `barbie`. Interestingly, this file is just an ordinary file.

Certain system components on a typical UNIX system look after delivery of (electronic) mail. In-coming mail is simply appended onto the `/var/mail/<username>` file for the specified user (the specified recipient).

When a user goes to read his or her mail, the mail user interface program (e.g. *pine*, *elm*, *mail*, *xmh*, *exmh*) will read the contents of `/var/mail/<username>` to obtain any new mail for that user. After successfully obtaining the new mail, the mail interface will truncate the `/var/mail/<username>` file.

Consider now the *finger(1)* command. When invoked with a username as an argument, it provides various information about that user. One piece of information it can provide relates to new mail for that user (mail that has been delivered into `/var/mail/<username>`). In particular, if the user has unread mail it can output the messages such as

```
New mail received <date-and-time-1>;
unread since <date-and-time-2>
```

where the two *date-and-time* terms are the corresponding dates and times.

How can the *finger(1)* program determine whether or not a user has new mail, when the last mail was received (delivered), and how long it has been since the user has read his or her mail? Further, how can it determine all this with a minimum amount of processing? I.e. what is a general algorithm for the simplest way for *finger(1)* to provide the functionality described above?

I. (10 marks)

The system library call *gethostbyname(3)* returns a pointer to a statically allocated hostent structure. Thus if *gethostbyname()* is called multiple times, the results of the previous *gethostbyname()* call in the statically allocated memory locations will be overwritten. This is problematic if one wants to collect host information for a number of machines prior to making use of all the host information.

One way to deal with this problem is to copy the `hostent` structure to dynamically allocated memory. However, the structure contains pointers to statically allocated information, and arrays of pointers to statically allocated information. Instead of copying the array of pointers, one needs to copy the pointed-to information, and then create a new array of pointers.

Write a function, *hostentdup()*, which will duplicate a `hostent struct` (e.g. as returned by *gethostbyname(3)*) into a dynamically allocated memory. The function should accept a pointer to a `hostent struct` as its input argument. On success, it is to return a pointer to the duplicate `hostent struct`, and on failure it should return NULL. Specify any "`include` files" which are required by your function.

Excepts from the man page for *gethostbyname()* are at the end of this exam booklet.

Portions of *man* pages from various system and library calls which you may find useful are given below.

```
FGETS(3)                  NetBSD Programmer's Manual                  FGETS(3)

NAME
     fgets, gets - get a line from a stream

SYNOPSIS
     #include <stdio.h>

     char *
     fgets(char * restrict str, int size, FILE * restrict stream);

     char *
     gets(char *str);

DESCRIPTION
     The fgets() function reads at most one less than the number of characters
     specified by size from the given stream and stores them in the string
     str.  Reading stops when a newline character is found, at end-of-file or
     error.  The newline, if any, is retained.  In any case a `\0' character
     is appended to end the string.


     The gets() function is equivalent to fgets() with an infinite size and a
     stream of stdin, except that the newline character (if any) is not stored
     in the string.  It is the caller's responsibility to ensure that the in-
     put line, if any, is sufficiently short to fit in the string.

RETURN VALUES
     Upon successful completion, fgets() and gets() return a pointer to the
     string.  If end-of-file or an error occurs before any characters are
     read, they return NULL.  The fgets() and functions gets() do not distin-
     guish between end-of-file and error, and callers must use feof(3) and
     ferror(3) to determine which occurred.
```

```
STAT(2)                   NetBSD Programmer's Manual                   STAT(2)

NAME
     stat, lstat, fstat - get file status

The status information word st_mode has the following bits:

     #define S_IFMT 0170000          /* type of file */
     #define        S_IFIFO  0010000  /* named pipe (fifo) */
     #define        S_IFCHR  0020000  /* character special */
     #define        S_IFDIR  0040000  /* directory */
     #define        S_IFBLK  0060000  /* block special */
     #define        S_IFREG  0100000  /* regular */
     #define        S_IFLNK  0120000  /* symbolic link */
     #define        S_IFSOCK 0140000  /* socket */
     #define        S_IFWHT  0160000  /* whiteout */
     #define S_ISUID 0004000  /* set user id on execution */
```

```
#define S_ISGID 0002000  /* set group id on execution */
#define S_ISVTX 0001000  /* save swapped text even after use */
#define S_IRUSR 0000400  /* read permission, owner */
#define S_IWUSR 0000200  /* write permission, owner */
#define S_IXUSR 0000100  /* execute/search permission, owner */
#define S_IRGRP 0000040  /* read permission, group */
#define S_IWGRP 0000020  /* write permission, group */
#define S_IXGRP 0000010  /* execute/search permission, group */
#define S_IROTH 0000004  /* read permission, other */
#define S_IWOTH 0000002  /* write permission, other */
#define S_IXOTH 0000001  /* execute/search permission, other */
```

GETHOSTBYNAME(3) NetBSD Programmer's Manual GETHOSTBYNAME(3)

NAME

gethostbyname, gethostbyname2, gethostbyaddr, sethostent, endhostent, herror, hstrerror - get network host entry

SYNOPSIS

```
#include <netdb.h>
extern int h_errno;

struct hostent *
gethostbyname(const char *name);
```

DESCRIPTION

The gethostbyname(), gethostbyname2() and gethostbyaddr() functions each return a pointer to an object with the following structure describing an internet host referenced by name or by address, respectively. This structure contains either the information obtained from the name server, named(8), broken-out fields from a line in /etc/hosts, or database entries supplied by the yp(8) system. The order of the lookups is controlled by the `hosts' entry in nsswitch.conf(5).

```
struct  hostent {
        char    *h_name;        /* official name of host */
        char    **h_aliases;    /* alias list */
        int     h_addrtype;     /* host address type */
        int     h_length;       /* length of address */
        char    **h_addr_list;  /* list of addresses from name server */
};
#define h_addr  h_addr_list[0]  /* address, for backward compatibility */
```

The members of this structure are:

h_name — Official name of the host.

h_aliases — A NULL-terminated array of alternative names for the host.

h_addrtype — The type of address being returned; currently always AF_INET.

h_length — The length, in bytes, of the address.

h_addr_list — A NULL-terminated array of network addresses for the host. Host addresses are returned in network byte order.

```
     h_addr      The first address in h_addr_list; this is for backward com-
                 patibility.

DIAGNOSTICS
     Error return status from gethostbyname(), gethostbyname2() and
     gethostbyaddr() is indicated by return of a null pointer.  The external
     integer h_errno may then be checked to see whether this is a temporary
     failure or an invalid or unknown host.  The routine herror() can be used
     to print an error message describing the failure.  If its argument string
     is non-NULL, it is printed, followed by a colon and a space.  The error
     message is printed with a trailing newline.

BUGS
     These functions use static data storage; if the data is needed for future
     use, it should be copied before any subsequent calls overwrite it.  Only
     the Internet address format is currently understood.
```

```
MALLOC(3)                 NetBSD Programmer's Manual                 MALLOC(3)

NAME
     malloc, calloc, realloc, free - general purpose memory allocation func-
     tions

SYNOPSIS

     #include <stdlib.h>

     void *
     malloc(size_t size);

     void *
     calloc(size_t number, size_t size);

     void *
     realloc(void *ptr, size_t size);

     void
     free(void *ptr);
```

Epilog

That's it! You're all done!

I sincerely hoped that you learned a lot in this course, and will find the material covered in the course valuable in your future endeavors.

Given all you've learned in the course, I am confident that you will be able to interpret the meaning of the following shell script, which typically bounces around the Internet this time of year:

```
better [ !pout -a !cry ]
better [ !shout ]
cat /etc/why
santa_clause < north_pole > town

cat /etc/passwd | awk 'BEGIN {FS=":"} {print $1}' > list
/usr/bin/check list
/usr/bin/check list
cat list | sgrep naughty > /dev/coal
cat list | sgrep nice > /dev/presents
santa_clause < north_pole > town

who | sgrep sleeping
who | sgrep awake
who | sgrep bad
who | sgrep good
for goodness sake; do
   be good
done

better [ !pout -a !cry ]
better [ !shout ]
cat /etc/why
santa_clause < north_pole > town
```

Extra Space

(The space below is for answering previous questions or for rough work.)